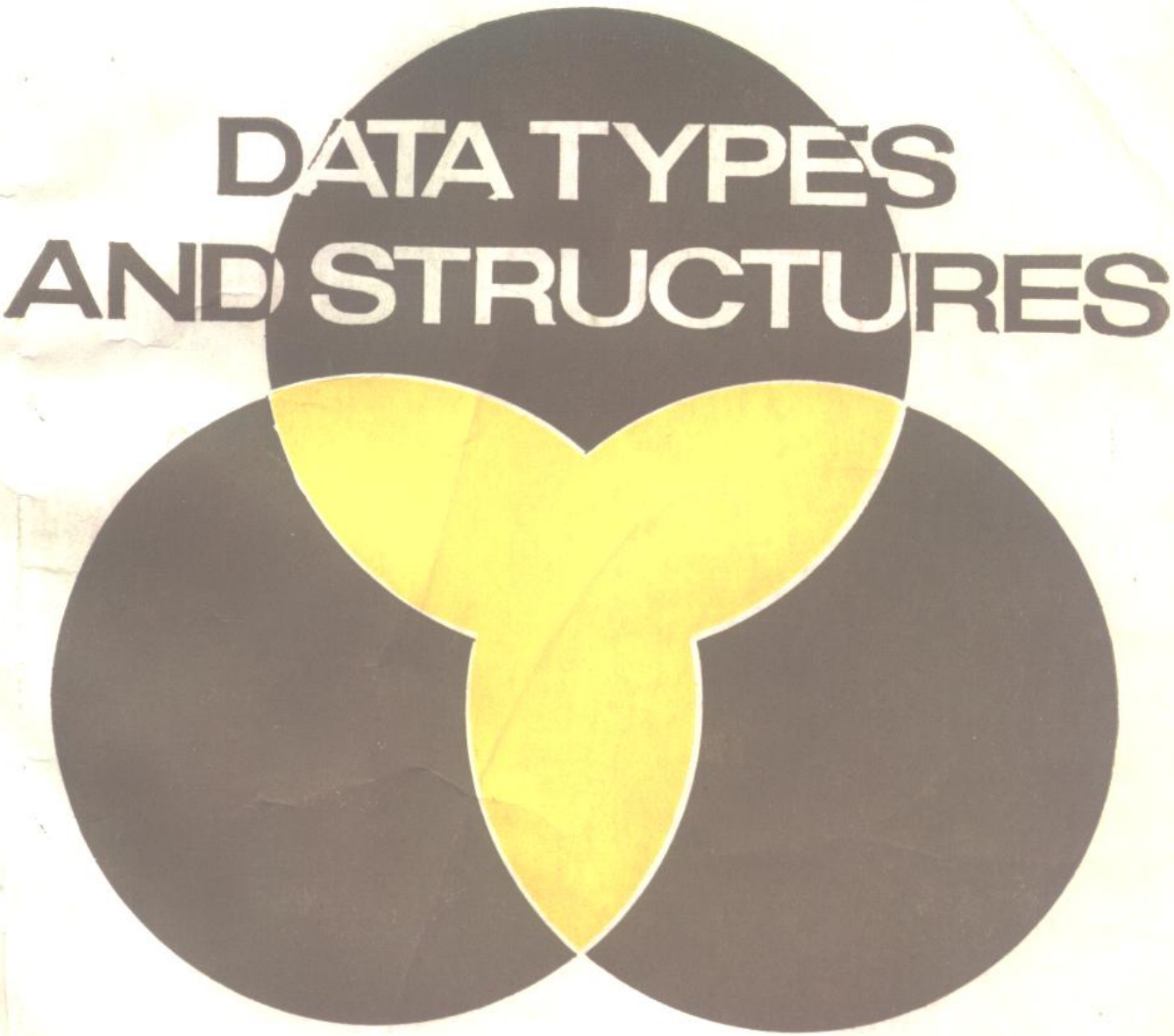
数据类型与结构

〔加拿大〕C. C. 高脱里
Leo R. 高脱里 著



机 械 工 业 出 版 社

数 据 类 型 与 结 构

〔加拿大〕C. C. 高脱里
Leo R. 高脱里 著

张延炬 张国衡 朱亦梅 译
郝忠孝 王汝范 校

机 械 工 业 出 版 社

DATA TYPES AND STRUCTURES
C.C.Gotlieb LeoR.Gotlieb
PRENTICE-HALL 1978
数据类型与结构
〔加拿大〕 C.C.高脱里 LeoR.高脱里 著
张延炬 张国衡 朱亦梅 译
郝忠孝 王汝范 校
*
机械工业出版社出版(北京阜成门外百万庄南街一号)
(北京市书刊出版业营业许可证出字第117号)
机械工业出版社印刷厂印刷
新华书店北京发行所发行·新华书店经售
*
开本 787×1092 1/16·印张 21·字数 510 千字
1985年9月北京第一版·1985年9月北京第一次印刷
印数 00,001—10,100·定价 5.00元
*
统一书号: 15033·5782

序

数据结构在美国计算机协会教程 68 (ACM's Curriculum68)① 中已被确定为一门与众不同的学科，普遍认为它在高等学校教学大纲中是计算与信息科学(专业)方面第一门有系统的、综合性的课程。但是在教程 68 中该学科的范围并没有明确规定。自教程 68 以来，所有涉及"数据结构"这一术语的论文、评述和书籍都促使这门课程逐渐建立了完整的理论基础。最初，数据结构几乎和图论是同义的，特别是关于表和树的理论，它们很自然地适合于描述分层数据。于是，这个概念扩展到包括网络、集合代数、关系、格等等，形成了目前**离散结构** (discrete structure) 的内容。随着"Codasyl Task Force"刊物的出版，并且通过**存贮结构** (storage structure) 的处理，使数据结构的数学概念得到了进一步的扩展，对于数据结构在计算机存贮中的描述，目前主要不在于数学水平的高低。美国计算机协会召开的"数据：概念、定义与结构"会议进一步加深了对这门学科的理解。在那次会议中，有几篇关于数据类型的论文，其概念曾出现在 Algol 的第一个文本中，而在此之前却没有引起人们的注意。

至于数据结构的分支——"离散代数结构"，仍然是通过考察数据的集合及对它们的运算；通过这些运算对数据类型的分类；以及通过在计算机语言中确定类型并在应用中使用这些类型的设计方法等等去进行研究的。我们关心的是数据结构的主体部分即存贮结构，因此第 2 章为本书提供了一个提纲，包括存贮结构的定义。同时，通过后边的章节反复提出由存贮表示法带来的问题并给出解决的方法；例如，在第 5 章中数组的表示法，在第 8 章中存贮的分配与再分配问题等等。因为计算机存贮对于数据结构是如此重要，所以计算机硬件方面的课程应是数据结构课程的前驱课，本书则是作为数据结构课的教材而编写的。在学习本课程前，"离散结构"知识是必不可少的，所以在第 1 章中先介绍一些必要的预备知识。因为，虽然在大多数计算机科学（专业）的教学大纲中，"离散结构"课被当作前驱课，但也有些学校是把它与"数据结构"课同时进行的。

在计算与信息科学（专业）的教学大纲中，"数据结构"课程的位置非常合适：它刚好设在计算方面的入门课程之后，这些课程通常主要是解释算法的概念，开发程序设计的风格、样式，并向学生介绍一种或多种程序设计语言；它又刚好设在详细说明编译理论、操作系统及文件系统方面的课程之前，它体现了这一学科的大纲。所以，数据结构课是为计算机科学、信息科学、系统科学（各专业）以及希望在诸如应用数学或工业工程等有关领域中进一步得到一些计算机程序设计知识的学生而开设的。确实，某些高超技术的产生，诸如本书中讨论的分类与无用单元收集方面的工作程序，似乎仅仅是专家的事情。但这些内容是如此普遍而又如此重要，因此理解其原理是任何从事程序设计者都应具备的基本知识。本书讨论的就是其中最基本的内容。

在第 2 章中介绍了数据类型与数据结构的基本概念，它为以后的章节搭了一个骨架；第 3 章到第 7 章详细研究了一般类型、串、表、数组、树、集合与图。除了这些基本原理以外，本书还打算提出一些我们认为十分重要的思想。这些思想可以归纳如下：

● 数据类型的定义是与程序语言的设计有密切联系的。许多程序设计语言都有其效率高、对某些特殊数据类型的开发利用卓有成效等优点，在数据结构课中使用这样的语言及围绕着它们提出的例子与问题，对于教学来说是合适的，也是令人感兴趣的。根据这个理由，本书

① 美国计算机协会1968年制定的一个教学方面的文件——译注。

中的算法都不仅仅是采用单一的语言。本书曾采用了一种简单的类似于 Algol 的语言，它有足够的符号去选择字段并用指针变量进行工作。在编写中，我们尽力将各个算法表示成现今可以接受的形式，但对程序设计的风格、样式却没有去详加说明。所以期望读者能将算法重新加工成实际的语言，如 APL，SNOBOL，PASCAL 或 PL/I 等，以显示语言中的数据结构怎样有助于这一过程的实现。从这一点出发，有时我们也将个别算法用上述某一种语言写出。

● 多数最近开发的程序设计语言都考虑了允许用户引进新的适于应用的数据类型。这就自然地导致了选择与设计数据结构的问题。虽然这个过程是解决问题的一部分，但又没有手册可供利用，于是出现了建立、选择数据类型与数据结构的方法。在第 7 章中详细地说明了这些方法。为了使读者有系统地认识这些方法，本书从第 2 章到第 7 章列举了在相同的情况下比较与选择数据结构时所采用的许多准则。此外，在附录中还给出了数据结构设计的实例。

● 在涉及庞大的文件和数据库文件集合时，还会遇到描述与表示数据的很多问题。文件结构与数据库管理系统本身都是一门学科，在这些方面另有一整套书籍。但是读者要了解这些原理与概念，就要正确地了解一些比较简单的文件结构。因为数据量比较大，存贮映射涉及多级存贮，并且能把基本类型合并成组合类型，所以文件结构及数据库管理系统要更为复杂一些。但其基本的结构还是一些常见的基本类型，在任何严格的论述中阐明这一点是很重要的。因而第 9 章与第 10 章试图在数据结构、文件结构和数据库管理〔后者一般在计算机科学(专业)的课程中加以研究〕与定向学科之间提供联系，并为这些学科的后续课程打下基础。

● 即使数据结构的许多方面都是以数学（尤其是代数、图论及组合学）为基础的，但对另一方面（以程序设计语言作为核心）却尚未定形，因而该学科仍在迅速发展。在第 2 章中介绍了当前发展中的某些情况，另一些则在后边的章节中加以论述。同样，为提出数据结构的发展方向，我们在书中还介绍了联合存贮及数据操作语言等课题。虽然后边的一些章节与早期本课程内容相比，更多的内容已有了变化，我们认为本书已充分地包含了高等学校中本门课程应有的全部课题。

本书是加拿大多伦多大学等高等学校数据结构课程的教材，是计算机科学（专业）三年级的主干课。在这一方面，学生们还将学习一门算法与结构程序设计方面的入门课程，一门程序设计语言（学习三至四种语言，包括 PL/C，Algol W及 SNOBOL）和一门计算机结构学方面的课程。他们还要体验一下用几种结构类型去进行工作，但不是系统地概括这些类型。数据结构课对于以后的大多数专业课来说，是必学的前驱课。因为大多数学生还没有学过离散结构课，所以一个学期通常只讲授到第 7 章中间的一些主要内容为止，而把后边的章节挪到“文件结构与数据库”这门课程中。全书包括两个学期的教学内容。脚注中的文献资料与参考引证是为读者深入钻研而提供的。

作者认为，解题与做练习是任何课程都必不可缺少的；本书的每一章都给出了例子与习题。这些习题的数学阐述与解答都是重要的。但更重要的是对于概念的透彻理解，所以读者要区别了解每一个数据类型以及该类型所适用的场合。尤其希望读者在系统设计与计算中知道怎样使用定量的方法。许多习题都意图测验读者的理解程度并促使他们精通关于数据结构选择的定量比较方法。

作者对为本书作出贡献的专家和学者谨致衷心的谢意。（下略）

C．C．高脱里

LeoR．高脱里

目 录

第 1 章 数学预备知识 …… 1
1.1 集与数理逻辑 …… 1
1.2 关系 …… 3
1.3 函数、映射与算子 …… 6
1.4 串与文法 …… 8
1.5 图 …… 10
1.6 树 …… 12
1.7 有向图 …… 15
1.8 算法的时间特性 …… 17
习题 …… 18
参考文献 …… 20
第 2 章 基本概念和定义 …… 22
2.1 结构与处理 …… 22
2.2 整型数据 …… 24
2.3 原始型数据的模型 …… 26
2.4 复合型数据 …… 28
2.5 各种类型的算子 …… 33
2.6 类型说明与校验 …… 36
2.7 存贮映射 …… 39
2.8 数据结构评价 …… 46
习题 …… 47
参考文献 …… 48
第 3 章 串 …… 50
3.1 串的基本表示法 …… 50
3.2 串变量 …… 53
3.3 SNOBOL 中的串处理 …… 58
3.4 串匹配 …… 63
3.5 数据紧缩 …… 68
习题 …… 73
参考文献 …… 75
第 4 章 简单表 …… 77
4.1 表的类型与表示法 …… 79
4.2 具有可控访问点的表 …… 84
4.3 简单链表 …… 90
4.4 用程序设计语言处理表 …… 93
4.5 倒排表 …… 97
4.6 简单表的检索 …… 101
4.7 内部分类 …… 111
习题 …… 118
参考文献 …… 119
第 5 章 向量和数组 …… 121
5.1 数组的存贮映射 …… 121
5.2 数组的处理 …… 123
5.3 分块和稀疏数组 …… 130
5.4 分块和稀疏数组的存贮映射 …… 133
习题 …… 140
参考文献 …… 141
第 6 章 树型目录 …… 142
6.1 树型目录的种类 …… 143
6.2 树的操作 …… 147
6.3 位置树 …… 152
6.4 字典树 …… 159
6.5 约束树 …… 163
6.6 具有不等权结点的树 …… 172
6.7 混合树 …… 178
6.8 比较 …… 181
习题 …… 182
参考文献 …… 184
第 7 章 集合和图的结构 …… 185
7.1 集合的数据结构 …… 185
7.2 集合处理语言 …… 193
7.3 图的顶点-边-表表示法 …… 201
7.4 非环形图的应用 …… 205
7.5 广义图 …… 213
习题 …… 216
参考文献 …… 218
第 8 章 存贮管理 …… 220
8.1 一般策略 …… 220
8.2 分配和重新分配 …… 221
8.3 合并 …… 223
8.4 无用存贮单元的收集 …… 228
习题 …… 236
参考文献 …… 237
第 9 章 文件 …… 238

9.1 辅助存贮器的特性……238
9.2 文件格式……245
9.3 文件索引与目录……249
9.4 多属性检索……261
9.5 外分类……279
习题……290
参考文献……292
第10章 数据库模型 ……294
10.1 DBMS的概念 ……295
10.2 网络数据库管理系统 ……298
10.3 关系数据模型 ……306
习题……318
参考文献……319
附录 数据结构设计实例 ……321
A.1 访问通路和子结构 ……322
A.2 存贮要求 ……326
A.3 操作集 ……327

第1章　数学预备知识

关于数据结构的较精确的定义，我们将在第二章中介绍，而目前我们暂且只要认为它是指一些能存贮在电子计算机内，并被执行运算的数据集合就可以了。数据结构（*data structure*）和数学结构（*mathematical structure*）之间的区别并不十分明显，但数学基本上是研究结构和关系（*relation*）的抽象性质而不去考虑这些结构在某个设备中是如何表示的。在这一章中，我们将提供同研究数据结构有关的数学概念和符号。

1.1　集与数理逻辑

集（*set*）的原始定义就是一些元素或成员的集合。我们用下面的符号来表示集和有关集的简单属性，写成

$x \in X$　　表示 x 是集 X 的一个元素或成员；

$x \notin X$　　表示 x 不是集 X 的一个元素。

当元素在集中的排列次序有很重要的意义时，就应该明确地加以指出。我们用 $\{x_1, x_2, \cdots, x_n\}$ 表示元素 $x_1, \cdots, x_n$ 的**无序集**（*unordered set*）而 $(x_1, \cdots, x_n)$ 表示**有序集**（*ordered set*）。

有序集中一个元素的**下标**（*index*）是一个整数，它说明此元素在集中的位置。第一个元素的下标可以是0或者是1。符号 x_p 和 $x[p]$ 均可用来表示一个有序集中的第 p 个元素。

集的**基数**（*cardinality*）$|S|$ 表示该集中元素的总数。有两个重要的集：其一是**空集**（*nullset*），空集中没有元素，用符号 ϕ 表示；其二是给定集 A 的**势集**（*power set*），用符号 $P(A)$ 表示。$P(A)$ 是集 A 的所有**子集**（*subset*）的集合。若 $|A| = n$，则 $|P(A)| = 2^n$，因为 $P(A)$ 是由集 A 中每个元素或是被选取或者被删去而构成的。

任给两个集 A 和 B，若 B 中的任何成员都是 A 中的成员，我们说 A 含有 B 或者说 B 是 A 的一个子集，记作 $A \supseteq B$ 或 $B \subseteq A$。若 A 的成员中至少有一个不是子集 B 中的成员，则称 B 是 A 的**真子集**（*proper subset*）$(A \supset B)$。

$A = B$　表示集 A 和集 B 有相同的元素；

$A \cup B$　表示由集 A 或集 B 中所有的元素构成的集；

$A \cap B$　表示由集 A 和集 B 中的公共元素构成的集；

$A - B$　表示由存在于集 A 中，而不存在于集 B 中的元素所构成的集；

$\bar{A}$　表示由不存在于集 A 中而存在于某一**宇集**（*universal set*）$X \supset A$ 中的元素构成的集。

这些定义也可当作是运算符∪(并)，∩(交)，－(差)，以及¯(补)的定义。这里集是被运算的对象。

对于任意的集 A、B 和 C，算子∪和∩遵循某些定律，这可以由定义直接证明。它们是：

交换律：　　$A \cup B = B \cup A$ 以及 $A \cap B = B \cap A$

结合律：　$A \cup (B \cup C) = (A \cup B) \cup C$ 以及

$A \cap (B \cap C) = (A \cap B) \cap C$

分配律：　$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ 以及

$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$

两个算子∪和∩中的每一个都可分配到另一个中去。

例1.1

集的运算可用**文氏图**（*Venn diagram*）的平面图形来描述，图中的点用以表示集的元素。图1.1 a 是描述分配律的，

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

这里 $A \cap B \cap C \neq \phi$。类似地，图1.1 b 是定理

$$\overline{A \cup B} = A \cap B$$

的图解表示，此定理也可用运算符的定义来证明。在本书中，所有的集都是**无限可数的**（*countably infinite*），也就是说，它们的每一个元素都可以和一个整数相对应。因为平面上的点是不可数的，故用文氏图来图解集和集的运算是有意义的，即使集的元素为无限不可数时也是如此。

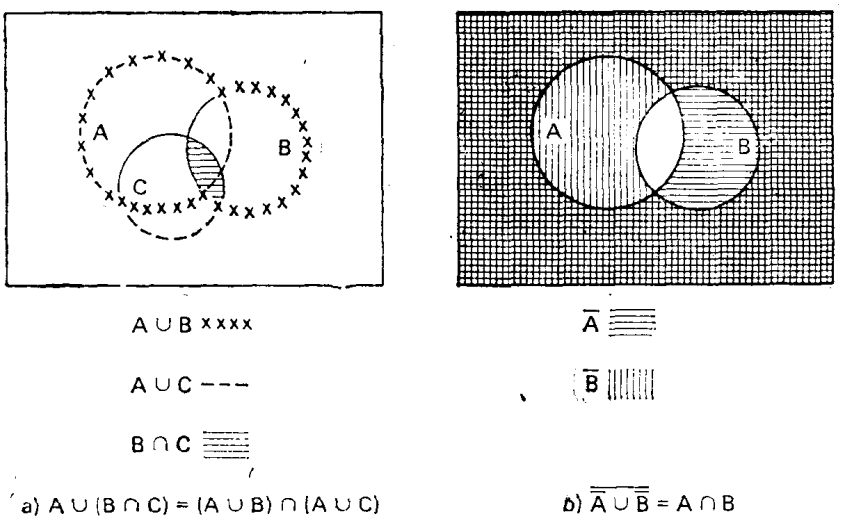


图1.1　集定理的文氏图解法

两个集的**笛卡尔乘积**（*Cartesian product*）$A \times B$ 是元素的**有序对**（或称序偶 *ordered pair*）的集。有序对中，头一个是集 A 中的成员，第二个是集 B 中的成员。

$$\{(a, b) \mid a \in A, b \in B\}。$$

此外，集的其他基本概念取自于数理逻辑。$X = \{x \mid p(x)\}$ 是当谓词 p 为真（即条件 p 成立）时，元素 x 的集。**布尔变量**（*Boolean variable*）的取值，有**真**（*true*）或**假**（*false*）两种（由1或0来表示），而对变量的布尔运算则有∧（与），∨（或）及⁻（非）。它们是用表1.1所示之真值表来定义的。

集与布尔变量之间以及集的运算与布尔运算之间是完全可以对应起来的。例如，对应于集的分配律，就有一个布尔变量的分配律：

$$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$$

及　　$A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$

表1.1　布尔算子的真值表

$A \wedge B$

A \ B	真	假
真	真	假
假	假	假

$A \vee B$

A \ B	真	假
真	真	真
假	真	假

$\bar{A}$

A	$\bar{A}$
真	假
假	真

1.2 关　系

两个集X和Y上的**二元关系**(*binary relation*)ρ是笛卡尔乘积$X \times Y$的一个子集。若$(x \in X,\ y \in Y)$是该子集的一个成员，我们记作$x\ \rho\ y$。作为特例，集X的二元关系则是$X \times X$(也可记作X^2)的一个子集，并标记为$(X,\ \rho)$。

如果设M为月份的集{一月，二月，……，十二月}，而S是季节的集{春，夏，秋，冬}，那末"M是S中的一个月份"这一关系可用表1.2来表示。$M \times S$的成员中属于这一关系者为真，而不属于者为假。

表1.2　月份和季节的关系

M	春	夏	秋	冬	M	春	夏	秋	冬
一月	假	假	假	真	七月	假	真	假	假
二月	假	假	假	真	八月	假	真	假	假
三月	真	假	假	真	九月	假	真	真	假
四月	真	假	假	假	十月	假	假	真	假
五月	真	假	假	假	十一月	假	假	真	假
六月	真	真	假	假	十二月	假	假	真	真

对于分别具有n和m个元素的**有限集**(*finite set*)X和Y，相应地可以用一个$n \times m$的矩阵来表示关系$(X,\ Y,\ \rho)$，这里矩阵的行和列分别与X和Y的元素相对应，若二个元素是相关的则表中记入1，否则则记入0。这是一个**与关系相对应的矩阵，称关系矩阵**。它是一个**布尔矩阵**(*Boolean matrix*)，它的元素取值为**真**或**假**。对于$(X,\ \rho)$，此矩阵为一方阵。表1.3列出了与关系({1，2，3，4，}<)相对应的矩阵。

表1.3　与一个关系相对应的矩阵

	1	2	3	4
1	0	1	1	1
2	0	0	1	1
3	0	0	0	1
4	0	0	0	0

值得注意的是确定某子集$R \subseteq X^2$的一个二元关系$(X,\ \rho)$同时也确定了一个**补集**(全集*complement*)$\bar{R} = X^2 - R$及与它相应的**补关系**(全关系*complementary relation*)$(X,\ \bar{\rho})$。

关系({1，2，3，4，5}，<)的补关系是({1，2，3，4}≮)，而它的成员正是表1.3中取0值的那些元素。

二元关系并非是可被用来表达集的元素之间关系的最普遍形式。例如，我们可以考虑以三个集的**三元关系**(*ternary relation*)来作为笛卡尔乘积$X \times Y \times Z$的子集。然而，就大多数问题而言，二元关系已足够用了，而且，还常常可把三元关系看成是由几个二元关系所组成的集。我们可以把与$X \times Y \times Z$关系相对应的三维矩阵看作是对于每一个$z \in Z$的X，Y平面的集，也即对于每一个Z，都有一个$X \times Y$的二元关系与之对应。(当然，也可把三元关系看成是对每一个$x \in X$的$Y \times Z$的二元关系的集，或对每一个$y \in Y$的$X \times Z$的二元关系的集)。

关系(X，ρ)有一些性质，我们称之为：

自反的(*reflexive*)：若对全体$x \in X$有$x \rho x$

非自反的(*irreflexive*)：若没有$x \in X$，有$x \rho x$

对称的(*symmetric*)：若对于全体x，$y \in X$有$x \rho y \Rightarrow y \rho x$

反对称的(*antisymmetric*)：若对于全体x，$y \in X$有$x \rho y \wedge y \rho x \Rightarrow x = y$

传递的(*transitive*)：若对于全体x，y，$z \in X$有$x \rho y \wedge y \rho z \Rightarrow x \rho z$

由自反/非自反，对称/反对称，和传递/不传递的各种不同组合而成的关系无论在生活中还是在数学中都是很有趣的，例如，兄弟关系(*sibling relationship*)，是非自反而对称的；关系"x是y的子结构"是非自反、反对称并且传递的。下面将要讲到的等价关系和半序，是特别重要的，因为它们是经常遇到的。

一个关系若是自反的、对称的和传递的，则称它是一个**等价关系**(*equivalence relation*)，用E来表示。一个集X的**分割**(*partition*)，是一簇子集X_1，……，$X_K \subseteq X$，用Π表示。这里满足下列条件：

1. 对所有的i，$X_i \neq \phi$(子集非空集)。
2. $i \neq j \Rightarrow X_i \cap X_j = \phi$(子集不相交)。
3. $\cup_i X_i = X$(各子集汇合起来构成X)。

给定X的分割Π，对于定义在X上的二元关系$E(\Pi)$(记作$x_i E(\Pi) x_j$意味着x_i和x_j相关的充要条件是x_i和x_j属于同一个子集。显然$E(\Pi)$是对称、传递和自反的，因此它是一个等价关系。不难证明，其逆也真，即：若一个集中存在一个等价关系必蕴含着一个分割。分割的一个子集$C(a)$，由符合条件$x E(\Pi) a$的全体成员组成：

$$C(a) = \{x \in X \mid x E(\Pi) a\} \subset X。$$

整数N中等价关系最普通的例子是以k(整数)为模的**剩余类**(*residue class*)组成的集。这是一个以N被k除后余留下来的整数之集，记作$N \bmod k$。每一个整数n均可写成$i \times k + m$的形式，这里i是一个整数且$0 \leqslant m < k$。这就使整数被分割为剩余类$\{C(0), C(1), \cdots\cdots, C(k-1)\}$。例如，如果模为7，则任何整数被7除后可能产生的余数是0，1，2，3，4，5，6，它们即代表了剩余类$C(0)$，……，$C(6)$。属于$C(1)$的整数就有$7 \times i + 1$的形式。

半序关系(*partial order relation*)$(X, \leqslant)$具有下列性质：

1. $x \leqslant x$(自反的)。
2. $x \leqslant y \wedge y \leqslant x \Rightarrow x = y$(反对称的)。

3. $x \leqslant y \wedge y \leqslant z \Rightarrow x \leqslant z$ (传递的)。

成员之间有半序关系的集称为**半序集**(*poset*)。我们可以从关系≤出发去定义其他的诸如≥、<和>等等关系①。例如：

$$x \leqslant y \Rightarrow y \geqslant x$$

$$x < y \Rightarrow x \leqslant y \wedge x \neq y$$

关系≥也是一个半序，但>（还有<）是非自反的因而不是一个半序。

注意，$x \not< y$并不蕴含$x < y$，因为<关系并不定义为≥关系之补。例如，假如≥是一个集中的**包含关系**(*inclusion relation*)；这是一个半序，而$X > Y$则被理解为Y是X的一个真子集。给出二个任意的集，我们不能说其中一个必定是另一个的子集。

如果对于任何二个元素x、y都有$x \geqslant y \vee x < y$，则称此为一**线性序**(*linear order*)或**单一序**(*simple order*)。此时半序集的成员可以用一条直线上的点来表示，而集则被称为**链**(*chain*)。

任何有限的半序集均可用一个**哈斯图**(*Hasse diagram*)来表示，其中：

1. 每一个各别的元素用一个点来代表。

2. 关系$x < y$用一条从x均匀上升到y的直线来表示。

在哈斯图中，若x在y的下面，并不一定表示$x < y$，除非在它们之间有一条连线。

例1.2

图 1.2 所示为半序集的哈斯图，它的元素就是相应的文氏图中代表子集的面积。此关系就是集的包含关系。

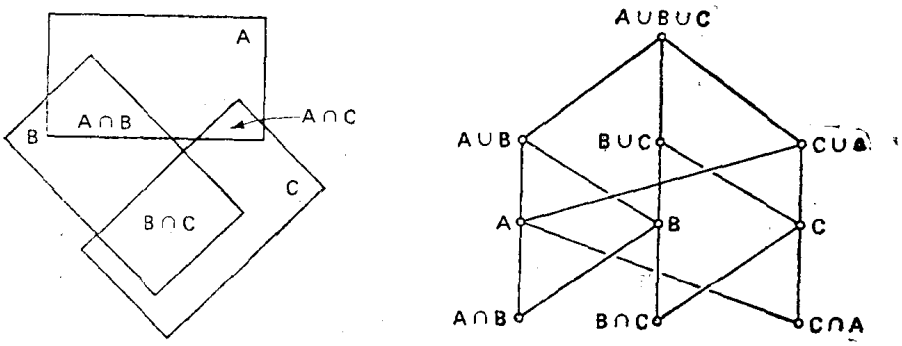


图1.2　哈斯图

令$(X, \leqslant)$是一个关系且$X \supseteq Y \supset \phi$。则当且仅当对所有的$y \in Y$，有$y \leqslant x$时，元素$x \in X$是$Y$的一个**上界**(*upper bound*)。所有Y的上界中最小的元素称为Y的**上确界**(*least upper bound*)(LUB)。同理也可类似地定义**下确界**(*greatest lower bound*)(GLB)。

格(*lattice*)是一个半序集，其中任意二个元素具有一个上确界和一个下确界。一个有限格的哈斯图通常在顶上和底部都只有一个点（见图1.3，图中画出了当关系为包含关系时，由势集$\{a, b, c, d\}$形成的格)。可以列举正整数的集$\{1, 2, \cdots\cdots, k\}$作为格的另

①有好几种用语句来表达这些关系的一般方法：例如，用x的队列少于或等于y表达$x \leqslant y$，用x先于y表达$x < y$，和用x是y的后继表达$x > y$。

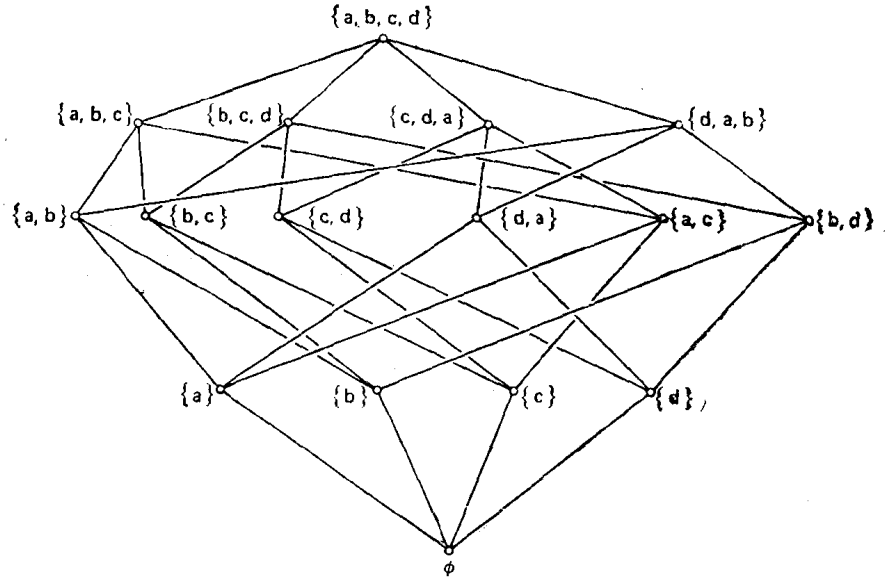


图1.3　势集{ a，b，c，d }的格

一个例子，关系为无余数的除法，所以 $a \geqslant b$ 表示 a 是 b 的一个倍数。此时上确界为最小公倍数，而下确界则是最大公因子。格的其他例子将在 2.4 节中给出。

1.3　函数、映射与算子

人们常用略有不同的术语来表达许多刚才已介绍过的相同的概念。若 X 和 Y 都是集，**函数**(*function*) f：$X \longrightarrow Y$ 是指这样的一个关系，即：若 xfy_1 并且 xfy_2 则必有 $y_1 = y_2$，这里 $x \in X$，而 y_1，$y_2 \in Y$。对于 $(x, y) \in f$ 我们写作 $f(x) = y$ 且称 x 为函数的**自变量**(*argument*)而 y 为函数的**值**(*value*)。函数与关系相比有一个重要的限制，即函数对于给定的自变量都必须是单值的。一个函数 f：$X \Rightarrow Y$，当且仅当对应每一个点 $y \in Y$ 都至少有一个点 $x \in X$ 使 $f(x) = y$ 成立，则称 f 为**映上的**(*onto*)或**满射**(*surjective*)。

集 X 称为函数 f 的**定义域**(*domain*)，而集 Y 则是函数 f 的**值域**(*range*)；y 称为在 f 下 x 的**象**(*image*)，而 $\{x \in X \mid f(x) = y\}$ 则称为 y 的**逆象**(*inverse image*)。记作 $x = f^{-1}(y)$。

一个函数 f：$X \Rightarrow Y$ 称为**一对一的**(*one-to-one*)或称作**内射**(*injective*)的充分和必要条件是：对于每一个点 $y \in Y$，若逆象 $f^{-1}(y)$ 存在，必仅由唯一的一个 x 值构成。这也就是说，只有在不同的点的象也不相同的时候，f 才称为内射。一个函数 f：$X \Rightarrow Y$，当它既是满射又是内射时，才称为**双射**(*bijective*)。对于一个双射而言，f 和 f^{-1} 都是函数。

二个函数 f 和 g 当且仅当 f 的值域等于 g 的定义域时，它们的**复合**(*composition*)才存在。此时，它由式 $h(x) = g(f(x))$ 给出。

有时候，希望用一个函数的术语来描述二个集 X 和 Y 之间的关系，甚至一个 $x \in X$ 值可能有几个值 y_1，……，y_n 与之对应的情况下也是如此。为此，可以定义一个关于给出子集 $X' \subseteq X$ 的**特征函数**(*characteristic function*) F。特征函数取值为 1 或 0（或真或假），这要根

据它的自变量是否是 X'的一个成员来决定；也就是说，若 $x \in X'$则 $F(x)=1$，而若 $x \notin X'$则 $F(x)=0$。有了特征函数，就可以用特征函数的集 $F_1(X, y_1)\cdots\cdots F_k(X, y_k)$来取代关系 (X, Y, p)，这里 $y_1\cdots\cdots y_k$ 是 Y 之可能值的集。F_1 的定义域是 X 集和布尔关系 $y=y_1$，它的值域是 1 或 0 的数值对。

例1.3

设零件号码与供应商之间的关系如表 1.4 所示。表中取值为 1 者表示它所处的这一行所表示的零件号码可以从它所处的这一列所表示的供应商那里得到供应。这一关系并不是一个函数，因为对于一个给定的零件号码可以有不止一个供应商。然而，关系的每一列都是零件号码和所对应的供应商间的一个特征函数。

表1.4 零件号码-供应商关系

零件号码	供应商		
	A	*B*	*C*
105-3	1	0	1
211-1	0	1	0
225-2	1	0	1
436-2	1	1	0

集 X 中的**二元运算**(*binary operation*)是一个从集 $X\times X$ 到集 Y 的一个函数，可以写成 $x_1\cdot x_2=y$，这里 x_1，$x_2\in X$ 而 $y\in Y$。假如这个函数是从 $X\times X$ 到 X 的，则运算称为**封闭的**(*closed*)。当且仅当对所有的 $x\in X$，$y\in Y$，及 $z\in Z$，若均有 $x\cdot y=y\cdot x$ 时，二元运算称为**可变换的**(*commutative*)，若均有 $(x\cdot y)\cdot z=x\cdot(y\cdot z)=x\cdot y\cdot z$ 时，二元运算称为**可结合的**(*associative*)。

二元运算中的**单位元素** (*identity element*) e 是这样的一个元素，即对于所有的 $x\in X$，必有 $x\cdot e=e\cdot x=x$。

半群 (*semigroup*) 是指 X 集连同 X 中存在的一个可结合的二元运算。带有单位元素的半群称为一个**独异点** (*monoid*)。一个**群** (*group*) 是这样的一个独异点：这里对每一个 $x\in X$ 元素均存在这样一个**逆元素** (*inverse element*) $x^{-1}\in X$，使得 $x\cdot x^{-1}=x^{-1}\cdot x=e$ (即单位元素)。

由集和关系来描述的系统也可用集和**算子** (*operator*) 来加以描述。这样一来，为了定义作为半序集的格 (X, p)(此时，任何二个元素都具有一个上确界和下确界)，我们可以定义二元算子 JOIN 和 MEET，使 xJOINy 和 xMEETy 分别对应于每对 x，y 的上确界和下确界。这么一来格可用 $(X$,JOIN, MEET$)$ 来表示。在势集格中 (关系是包含关系) JOIN 和 MEET 运算就是**并集** (*set union*) 和**交集** (*set intersection*)。

这一方法导至代数系统和抽象代数的产生。**(抽象) 代数** (*abstract algebra*) $\langle X, f_1, f_2, \cdots, f_k\rangle$ 是一个包括集 X 和算子号 $f_1\cdots f_k$ 的系统，这里 $f_i: X^{n_i}\longrightarrow X$ 是一个取 n_i 个自变量的算子。而重要的算子则是二元，一元和零元 (即 0 自变量) 算子 (也即常量)。

由于赋予代数的属性越来越多 (更多的算子，结合律，分配律，等等) 人们赋予代数系统的结构形式也越来越多，而抽象代数也就变成为人们熟悉的代数系统了。具有二元运算$\wedge$和$\vee$及一元运算 $^-$ (补) 的布尔代数以及对整数运算的整数域都是这方面的例子。在数据结

构的研究中，我们会遇到集和算子。通常，算子的种类和属性的数目不会象代数里研究的数学结构那样多。但是，我们仍要介绍一些性质、算子及其属性确定某种代数的概念，该概念在研究数据结构、程序设计语言和计算系统中很有用处。作为一个例子，我们将在下一节中讨论**串**（*string*）的代数。

1.4 串与文法

令$V=\{a, b, \cdots, z\}$为一个包含全部字母符号的集，V^*为包含全部字母串的集。V^*的成员是V的元素的有序集。

没有字符的串称**空串**（*null string*），用〃表示。**联接算子**（*concatenation operator*）Θ定义为：

$$\langle x_1 x_2 \cdots x_k\rangle \Theta \langle y_1 y_2 \cdots y_l\rangle = \langle x_1 \cdots x_k y_1 \cdots y_l\rangle。$$

显然Θ是符合结合律的，而〃起着单位元素的作用。（〃与由单个空白字符（*blank character*）构成的串是有区别的，后者标记为$\flat$）。因为

$$\langle x_1 \cdots x_k\rangle \Theta 〃 = 〃 \Theta \langle x_1 \cdots x_k\rangle = \langle x_1 \cdots x_k\rangle,$$

所以（V^*，Θ，〃）系统形成一个独异点。

同一个系统也可用关系来代替算子进行描述。为了代替联接算子，我们称$x=\langle x_1 \cdots x_k\rangle$为$y=\langle y_1 \cdots y_k \cdots y_l\rangle$的一个**前缀**（*prefix*）的充要条件是：$x_1=y_1$，$x_2=y_2, \cdots, x_k=y_k$且$k \leqslant l$。这一关系是自反的、反对称的、传递的，因此是一个半序。通常希望对串定义一个简单的序。为此，假定构成串的集中的符号有一个简单的序ρ_L。若$x\rho_L y$，我们就说x**先于**（*precede*）y。这称为对字符集的**对照序**（*collating sequence*）。为将ρ_L扩展到串中，令$\langle x_1 \cdots x_k\rangle$和$\langle y_1 \cdots y_l\rangle$都是串，这里$k \leqslant l$。若下列之一成立，则$\langle x_1 \cdots x_k\rangle \rho_L \langle y_1 \cdots y_l\rangle$

（a）$x_1 \rho y_1$或者

（b）$l=k$并且$\langle x_1 \cdots x_k\rangle = \langle y_1 \cdots y_l\rangle$或者

（c）对于$i=1$到$j<k$有$x_i=y_i$，$x_{j+1} \neq y_{j+1}$和$x_{j+1} \rho_L y_{j+1}$。

否则，$\langle y_1 \cdots y_l\rangle \rho_L \langle x_1 \cdots x_k\rangle$。如此定义的排序法正是众所周知的**词典顺序法**（*lexicographic order*）。再则，为取代串中使用的这一关系，可以定义一个**优先函数**（*precedence function*）Π，使其对于X，Y串有下面的关系：

当且仅当$X\rho_L Y$时，$X \Pi Y=$**真**。

实际上，辨别串相等，也就是它们一致的情况常常很有用。因此，我们可进一步规定优先函数Π，如下：

$$X \Pi Y = \begin{cases} \text{**少于**或 } -1 & \text{若} X\rho_L Y \wedge X \neq Y \\ \text{**等于**或 } 0 & \text{若} X = Y \\ \text{**大于**或 } 1 & \text{若} Y\rho_L X \wedge X \neq Y。 \end{cases}$$

由于串可被视为一个具有二元联接算子Θ和优先函数Π的集V^*，所以它们代表了抽象代数的一个例子。

串代数可以扩展到典型的程序设计语言中并在较小的程度上扩展到自然语言中。首先，前缀关系可推广到**子串**（*substring*）。其次，称为**生成式**（*production*）的算子的引入，使得以一个子串去替代另一个成为可能。生成式

$$\alpha\beta\gamma \Longrightarrow \alpha\beta\gamma$$

意味着 $\alpha\beta\gamma$ 串中的子串 β 被串 δ 所代替。

一个**短语结构文法**(*phrase structure grammar*) G，是一个四元集$\langle V_T, V_N, S, P\rangle$，这里

V_T = 给定的称为**终端符号**的集

V_N = 非终端符号的集

S = 指定的称为**初始**或**起始符**的非终端符号

P = 生成式的集。

句型 (*sentential form*) 是一个按生成式的次序由 S 导出的串；一个**句子** (*sentence*) 是由终端符号所组成的句型；G 的**语言** (*language*) 是句子的集合。文法和语言形式上可以当作生成式系统或者可以当作代数的一种类型来加以研究。

语言中重要的一类是用**与上下文无关的文法** (*context-free grammar*) 所产生的；这就是说，文法中每个生成式都具有这样的形式

$$U \Longrightarrow \delta$$

这里 U 是一个单个的非终端符号而 δ 是一个串。

例1.4

一种简单的与上下文无关文法是一种用作**标识符** (*identifier*)(例如，ALGOL60 中的变量名) 的串的生成程序。其组成部分是

$V_T = \{a, b, c, \cdots, z, A, B, \cdots, Z, 0, 1, \cdots, 9\}$

V_N = {IDEN, LETTER, DIGIT}

S = IDEN

集 P 由下列生成式构成：

LETTER $\Longrightarrow a \mid b \mid c \mid \cdots A \mid B \mid \cdots \mid Z$

DIGIT $\Longrightarrow 0 \mid 1 \mid \cdots \mid 9$

IDEN $\Longrightarrow$ LETTER | IDEN⊕LETTER | IDEN⊕DIGIT

生成式中垂直线符号 "|" 代表**交替算子** (*alternation operator*)，意思是算子的二边可以任意选择。

例1.5

与上下文无关文法的另一个例子是由变量的和、积以及用来指示表达式计算顺序的圆括弧一起所组成的算术表达式的生成。

$V_T = \{a, b, \cdots, z, +, -, *, \div, (,)\}$

V_N = {EXP, TRM, FCT, AOP, MOP, VAR}

S = EXP

生成式集 P 为

EXP $\Longrightarrow$ TRM | EXP AOP TRM

TRM $\Longrightarrow$ FCT | TRM MOP FCT

FCT $\Longrightarrow$ VAR | (EXP)

AOP $\Longrightarrow$ + | −

MOP $\Longrightarrow$ * | ÷

VAR $\Longrightarrow$ a | b | ⋯ z

要引导出句子，就要显示出从S导出句子的生成式的次序。下面就是对表达式（a + b）*（c + d）的推导，这是一个根据上面所定义的与上下文无关文法的一个句子。

EXP $\Longrightarrow$ TRM $\Longrightarrow$ TRM MOP FCT

TRM $\Longrightarrow$ FCT $\Longrightarrow$ (EXP) $\Longrightarrow$ (EXP AOP TRM)
$\Longrightarrow$ (TRM + FCT) $\Longrightarrow$ (FCT + VAR) $\Longrightarrow$ (VAR + b)
$\Longrightarrow$ (a + b)

MOP $\Longrightarrow$ *

FCT $\Longrightarrow$ (EXP) $\Longrightarrow$ (EXP AOP TRM) $\Longrightarrow$ (TRM + FCT)
$\Longrightarrow$ (FCT + VAR) $\Longrightarrow$ (VAR + d) $\Longrightarrow$ (c + d)

1.5　图

图是研究关系的另一种途径。**图**（*graph*）是由**顶点**（*vertex*）或**结点**（*node*）的一个（有限）非空集和一个**边**（*edge*）的集所组成，这里每条边是一个分离顶点的无序对。（图1.4就表示一个图）。一个图G的**阶**（*order*），是指图中顶点的数目，记作$|G|$。一条边决定了一对顶点之间的**关联关系**（*incidence relation*）。由一条边连接的二个顶点称为**相邻的**（*adjacent*）；否则，它们是**独立的**（*independent*）。若e是由结点x和y所确定的边（$e\sim(x,y)$），则称e与x，y相**关联**（*incident*）。该关系的唯一性质是对称性；也就是$(x,y)\in E\Longrightarrow(y,x)\in E$，这里$E$是边的集。

只具有一条边$e\sim(x,x)$也可定义为图。这样的边被称为**圈**（*loop*）。如果关系是自反的，那么每个顶点都有一个圈。也可能在给定的顶点对之间有多于一条的边相连。（见图1.5；在此情况下，并不强调把边解释为一个集，因为，配对并不明显）。不允许有多重边的图称为**简单图**（*simple graph*）。与顶点相关联的边的数目称为顶点的**度**（*degree*）。在一个**正则图**（*regular graph*）中所有顶点的度都是相同的。在n阶的**完全图**（*complete graph*）K_n中，每对结点都是相邻的（见图1.6，所示为多达K_6的完全图）。

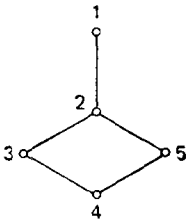


图1.4　图

V：{1，2，3，4，5}

E：{(1，2)，(2，3)，
(4，3)，(5，2)，(5，4)}

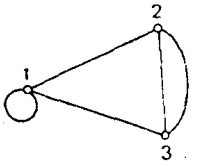


图1.5　带有圈和多个边的图

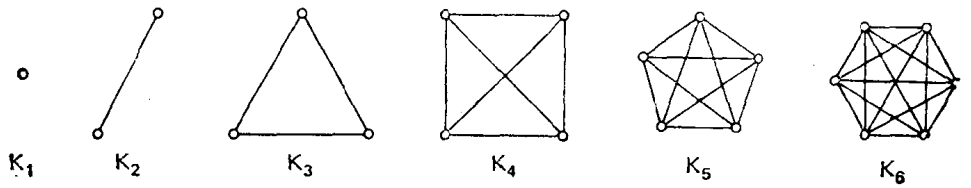


图1.6　多达K_6的完全图